Ukrainian Catholic University

Faculty of Applied Science

Eigenface-Based Face Recognition System

Semsichko Lidiya

Sampara Sofia

Bilyi Andrii

Mentor: Rostyslav Hryniv

Introduction

Face recognition is one of the key areas of computer vision and machine learning, with applications ranging from security systems and biometric authentication to social media and entertainment technologies. The fundamental idea is to employ mathematical and statistical methods to analyze and identify human faces within images.

The significance of face recognition has grown rapidly, as it is now embedded in everyday technologies — from unlocking smartphones and tagging friends in photos to enhancing surveillance and safeguarding banking operations. Its widespread adoption highlights both its practical value and the challenges associated with accurate and efficient identification.

A major milestone in the development of automated face recognition was the work of Sirovich and Kirby in 1987 [6]. They demonstrated that face images could be efficiently represented in a lower-dimensional subspace using Principal Component Analysis (PCA). Their research introduced the concept of representing faces as a combination of "eigenfaces," significantly reducing the complexity of facial data while preserving key distinguishing features. This approach was later extended by Turk and Pentland [7], who applied it directly to face recognition tasks, marking the beginning of modern statistical methods in this field.

Building upon these foundational ideas, this project aims to develop a user-friendly application that leverages PCA for face recognition. The system captures real-time images from a webcam, projects them into the eigenface subspace, and identifies individuals by comparing these projections against a predefined database. The application determines whether the person in the image is recognized (present in the dataset) or classified as unknown.

Possible approaches and related works

Face recognition has been extensively studied, with various methods developed over the years. These methods can generally be categorized into three main groups: geometric-based approaches, statistical methods, and neural network (NN) techniques.

Methods based on geometric characteristics [8], such as analyzing proportions and distances between facial features, were among the earliest approaches. While simple and computationally efficient, these methods tend to be sensitive to variations in head orientation, facial expressions, and lighting conditions, often leading to significant recognition errors.

In recent years, neural networks—particularly deep learning models—have become the state of the art in face recognition tasks due to their impressive accuracy and ability to generalize across complex variations [3]. Despite their success, these methods typically require large datasets and significant computational resources. Moreover, in the context of this project, which emphasizes linear algebra techniques, relying solely on NN-based solutions would deviate from the core objective. Nevertheless, to provide a comprehensive evaluation, we incorporated a neural network-based approach to compare its performance with PCA, particularly in real-time face recognition scenarios.

With advancements in statistical techniques, Principal Component Analysis (PCA) [5] emerged as a powerful tool for face recognition through dimensionality reduction. The classical eigenfaces approach, as explored in works like [1], applies PCA to transform high-dimensional face images into a lower-dimensional subspace by extracting uncorrelated features known as eigenfaces. Optimizations in such methods focus on reducing the number of principal components to improve computational efficiency while maintaining high recognition accuracy. For instance, experiments on datasets like face94 and ORL [11] demonstrate how PCA can effectively capture key facial variations and enable recognition by comparing distances between projected feature vectors in this reduced space.

Given these considerations, PCA was chosen as our primary method due to its strong theoretical foundation in linear algebra and its proven effectiveness in face recognition tasks, as supported by prior research.

PCA and other LA methods used in our project

Pros and cons of PCA

PCA reduces the dimensionality of the feature space and retains the most informative components. PCA also reduces the impact of noise and lighting variations, which improves recognition accuracy. However, PCA also has limitations: it is sensitive to alignment and scale, and its performance drops when faces are not centered or consistently illuminated. Despite these, PCA remains a practical choice for constrained environments. The choice of this approach is optimal for our task because it strikes a balance between efficiency, speed, and computational complexity.

Brief explanation of PCA

Every grayscale face image can be represented as a point in a high-dimensional vector space. For example, a 100×100 image corresponds to a vector in $\mathbb{R}^{10000}$. Working directly in such high-dimensional spaces is computationally expensive and often redundant, as many pixels carry correlated or insignificant information.

This raises a fundamental question: How can we represent faces efficiently while preserving essential features?

Principal Component Analysis (PCA) addresses this by finding a new coordinate system where data variability is maximized along the first few axes. Formally, PCA seeks an orthonormal basis — a set of eigenvectors — that captures the directions of maximum variance in the data.

Main steps of PCA-based face recognition

1. Prepare the Data. Flatten images into vectors and organize them into a matrix.
2. Center the Data. Subtract the mean vector (average face) from each image vector.
3. Compute the Covariance Matrix. Measure how features (pixels) vary together.
4. Calculate Eigenvectors and Eigenvalues. Find the directions (eigenvectors) where the variance is maximized, and their corresponding importance (eigenvalues).
5. Select Top Principal Components. Choose the eigenvectors with the largest eigenvalues to form a new subspace.
6. Project Data onto the New Subspace. Represent each face as a combination of the selected principal components (reducing dimensionality).
7. Use for Classification. Preprocess the test image (flatten and subtract the previously calculated mean) and project it onto eigenfaces. For classification, the Euclidean distance is calculated between the projected test image and the projections of all known faces in the subspace. The test image is assigned to the class of the closest projection or if the distance is bigger than the threshold - classify as unknown.

The theoretical part behind the algorithm with stress on the LA methods

1. Data Preprocessing and Representation

Assume we have a dataset of m grayscale images of size (h, w). Each image is reshaped into a column vector $\mathbf{x}_i \in \mathbb{R}^n$, where $n = h \cdot w$. In case of images, this n is huge. The goal of the algorithm is to reduce the dimensionality of the image vectors from n to a lower dimension k ($k < n$), where k is selected based on experimentation to retain sufficient variance while minimizing dimensionality of vectors (images).

All image vectors are combined into a data matrix:

$$A = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m] \in \mathbb{R}^{n \times m}$$

where each column represents a distinct image.

2. Data Centering

Before applying PCA, the dataset must be centered to have zero mean. The mean face vector is computed as:

$$\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i \in \mathbb{R}^n$$

Each image vector is then centered:

$$X = A - \bar{\mathbf{x}} \cdot 1_m^T$$

where 1_m is a vector of ones of size m. The resulting matrix X thus contains zero-mean pixel intensities, ensuring each pixel dimension has mean zero across the dataset.

3. Principal Component Analysis (PCA)

3.1 Covariance Matrix and Eigenfaces Computation

Firstly, we need to calculate the covariance matrix, which is defined as:

$$C = \frac{1}{m} X X^T \in \mathbb{R}^{n \times n}$$

Covariance measures how two variables change together. The covariance matrix captures how every pair of features (pixels of the images) co-vary. Thus, each element C_{ij} tells how feature i and feature j vary together.

The next step is to find the eigenvectors of the matrix C. The eigenvectors of the covariance matrix point in the directions where the data varies the most – these are the so-called principal directions. The corresponding eigenvalues tell us how much variance there is in those directions.

However, C is of size $n \times n$, which is huge in our case. So we use a trick.

We consider the smaller matrix:

$$C' = X^T X$$

which is of size $m \times m$.

We compute its eigenvectors by solving:

$$X^T X v = \lambda v$$

Multiplying both sides by X, we get:

$$XX^T(Xv) = \lambda(Xv)$$

So, the eigenvectors of C are obtained by left-multiplying the eigenvectors of C' by X.

We want to reduce the dimensionality to k components, so we need to find k dominant eigenvectors (kn), meaning we want to reduce the dimension of our vectors (images) from n to k elements. The dominant eigenvector is the one with the largest eigenvalue, since it captures the most variance in that direction. We use the Power Method [4] for this task.

3.2 Power Method and Deflation

Any arbitrary vector x_0 (except vectors orthogonal to v_1, the eigenvector of C' corresponding to the biggest eigenvalue, since in this case the coefficient near v_1 - c_1 - will be equal to zero and it will be impossible to find this eigenvector) can be expressed as a linear combination of eigenvectors of C':

$$x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

Now, apply C' repeatedly:

$$C' x_0 = c_1 \lambda_1 v_1 + c_2 \lambda_2 v_2 + \cdots + c_n \lambda_n v_n$$

$$C'^i x_0 = c_1 \lambda_1^i v_1 + c_2 \lambda_2^i v_2 + \cdots + c_n \lambda_n^i v_n$$

Because λ_1^i dominates all other terms (since $|\lambda_1| > |\lambda_2|$), the vector v_1 increasingly aligns with v_1, the dominant eigenvector, which allows the identification of v_1.

Also, to prevent numerical overflow or underflow during iterations, it is essential to normalize (rescale) the vector at each step:

$$x_{k+1} = \frac{C' x_k}{\|C' x_k\|}$$

This normalization ensures that the vector maintains a stable magnitude while converging to the direction of v_1 (or its deflated versions in subsequent computations).

The Power Method gives us only one eigenvector – the one corresponding to the largest eigenvalue λ_1. But we want to find k such vectors. So, we must modify the matrix to "remove" the influence of v_1, so that v_2 becomes the new dominant eigenvector. This process is called deflation.

We begin by writing the spectral decomposition of the covariance matrix C':

$$C' = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_n v_n v_n^T$$

To proceed with deflation, we remove the component associated with the dominant eigenvalue λ_1 and its corresponding eigenvector v_1. This gives us the deflated matrix:

$$C_1' = C' - \lambda_1 v_1 v_1^T$$

This operation subtracts the rank-1 matrix that captures all the variance in the direction of v_1, effectively setting the first term of the decomposition to zero. As a result, the new matrix C_1' starts its spectral decomposition from the second eigenpair λ_2, v_2, containing only the variance orthogonal to v_1. This process continues iteratively, removing one principal component at a time.

After finding all k eigenvectors of C', we form a matrix U', where the columns are these vectors. To return to our original covariance matrix $C = XX^T$, we multiply each found eigenvector by X and get the matrix columns of which are the eigenfaces:

$$U = XU'$$

However, since this multiplication does not preserve the norm, we must normalize each resulting column of U. This is because:

$$\|Xv\|^2 = v^T X^T X v = \lambda \|v\|^2$$

which shows that the resulting vectors Xv (the eigenfaces) will have magnitude scaled by both λ and $\|v\|$. Thus, to ensure that the eigenfaces form an orthonormal basis, each column of U must be normalized after projection:

$$u_i = \frac{Xv_i}{\|Xv_i\|}$$

This gives the matrix $U \in \mathbb{R}^{n \times k}$, whose columns are the normalized eigenfaces of the covariance matrix C.

4. Projection onto Eigenface Space

Once we have computed the eigenfaces, we obtain the matrix $U \in \mathbb{R}^{n \times k}$, whose columns form an orthonormal basis for the PCA subspace $L \subset \mathbb{R}^n$. Each column corresponds to one of the top k eigenvectors of the original covariance matrix C.

To represent a centered image vector $x \in \mathbb{R}^n$ in the eigenface basis, we compute:

$$y = U^T x$$

Here, $y \in \mathbb{R}^k$ contains the coordinates of x with respect to the basis formed by the eigenfaces. This is a change of basis operation. Each component y_i represents how strongly x aligns with the i-th eigenface.

For a dataset of m centered images stacked in the matrix $X \in \mathbb{R}^{n \times m}$, we compute low-dimensional representations: $Y = U^T X \in \mathbb{R}^{k \times m}$

This operation transforms each high-dimensional image into a compact representation of k coefficients. These coefficients capture the most significant variations in the dataset and serve as compressed facial descriptors in the PCA subspace.

5. Face Recognition via Nearest Neighbor

From the previous steps we have an eigenfaces basis and the projected training data images onto this space expressed as k-dimensional vectors y_i.

Assume, we have taken an image of some person and our goal is to identify whether this person is one of those from the dataset or unknown.

Given a new test grayscale image, we make a vector of its pixels data - $\mathbf{x}_{\text{test}}$ - and do the following:

1. Center the image the same way we did with the training dataset:

$$a_{\text{test}} = x_{\text{test}} - \bar{x}$$

2. Find the coefficients of this vector in the eigenface space we calculated before:

$$z_{\text{test}} = U^T a_{\text{test}}$$

3. Compute Euclidean distances to all training projections:

$$d_i = \left\| z_{\text{test}} - z_i \right\|_2$$

4. Reconstruct the test vector back into image-space and compute reconstruction error:

$$\widehat{a}_{\text{test}} = \bar{x} + U\, z_{\text{test}} \quad , \quad r = \left\| a_{\text{test}} - \widehat{a}_{\text{test}} \right\|_2$$

   If $r > T_r$, immediately classify as "Unknown."

5. Otherwise, take the label of the closest projection: if $\min_i d_i < T_d$, return its label; else return "Unknown."

Methodology and Pseudocode

In this section, we break down the face recognition procedure and present it in a clear algorithmic form. We have two main stages: (1) **PCA training** to compute eigenfaces from the training set, and (2) **Face recognition** for new images using the PCA representation and nearest-neighbor classification with thresholding.

PCA Training Algorithm

Below is a high-level pseudocode for computing the top k eigenfaces from a training set of face images. This assumes we have m training face images of size n (number of pixels), and we want the first k principal components. We use the power iteration an a loop to find each eigenvector one by one, deflating the covariance matrix after each is found.

```
Input: Flattened training images matrix $X \in \mathbb{R}^{n \times m}$ (each column is a centered face vector),
    number of components $k$
Output: Eigenfaces $\{u_1, ..., u_k\}$
 1: Compute the small covariance matrix: $S = X^T X \in \mathbb{R}^{m \times m}$
 2: Initialize eigenvector list $V = [\,]$
 3: for $j = 1$ to $k$ do
 4:     Choose a random vector $q$ of length $n$ (unit norm)
 5:     repeat
 6:         $q_{\text{next}} \leftarrow Sq$
 7:         Normalize $q_{\text{next}}$ to unit length
 8:     until $\|q_{\text{next}} - q\|$ is below tolerance
 9:     $v_j \leftarrow q_{\text{next}}$
10:     Append $v_j$ to $V$
11:     Deflation: $S \leftarrow S - (v_j (S v_j)^T)$
12: end for
13: Project eigenvectors back to high-dimensional space:
14: for $j = 1$ to $k$ do
15:     $u_j \leftarrow X v_j$
16:     Normalize $u_j$ to unit length
17: end for
18: return Eigenfaces $\{u_1, ..., u_k\}$
```

The algorithm takes as input a matrix of flattened training images, where each column corresponds to a centered face vector.

The process begins by computing the mean face, the average pixel vector in all training images, which is then subtracted from each image to center the data set around the origin. As discussed earlier, directly computing eigenvectors of the full covariance matrix $XX^T \in \mathbb{R}^{n \times n}$ is computationally expensive due to the large value of m compared to n. To address this, we construct the small covariance matrix $S = X^T X \in \mathbb{R}^{m\times m}$ in Step 3.

Next, to find the top k eigenvectors, we perform power iterations. In each iteration, we initialize a random unit vector and iteratively apply the power method to converge to the dominant eigenvector—the one associated with the largest eigenvalue in magnitude. Since we require multiple eigenvectors, we apply deflation after each one is found. This involves subtracting a rank-1 matrix (formed from the found eigenvector) to eliminate its influence from the covariance matrix.

Finally, the eigenvectors obtained from the small covariance matrix are projected back into the high-dimensional space to yield the corresponding eigenfaces, which serve as the principal components in the original image space.

Face Recognition Algorithm

After training, recognizing a new face involves projecting it into the PCA space and then using a nearest-neighbor search with a threshold. Pseudocode for recognizing a single test face is given below:

```
**Input:** Test image $x_{\text{test}}$; mean face $\mu$; eigenfaces $\{u_1, ..., u_k\}$;
    training feature vectors $\{z_i\}$ with labels $\{L_i\}$;
    distance threshold $T_d$; reconstruction threshold $T_r$
**Output:** Identity label of $x_{\text{test}}$ or “Unknown”
  Flatten $x_{\text{test}}$ into a vector and center it:
    $a_{\text{test}} \leftarrow x_{\text{test}} - \mu$
  Project into PCA subspace:
    $z_{\text{test}} \leftarrow [\, v_1^T a_{\text{test}}, \ldots, v_k^T a_{\text{test}} \,]$
  Reconstruct back into face-space:
    $\widehat{a}_{\text{test}} \leftarrow \mu + \sum_{j=1}^{k} v_j \,(v_j^T a_{\text{test}})$
  Compute reconstruction error:
    $r \leftarrow \| \, a_{\text{test}} - \widehat{a}_{\text{test}} \, \|_2$
  Initialize $min\_distance \leftarrow \infty$, $best\_label \leftarrow$ None
  for each training feature vector $z_i$ with label $L_i$ do
      Compute distance: $d \leftarrow \| \, z_{\text{test}} - z_i \, \|_2$
      if $d < min\_distance$ then
          $min\_distance \leftarrow d$
          $best\_label \leftarrow L_i$
      end if
  end for
  if $min\_distance < T_d$ and $r < T_r$ then
      return $best\_label$
  else
      return “Unknown”
  end if
```

In this recognition stage, we begin by centering the incoming test image x_{test} using the same mean face μ computed during training. We also compute the reconstruction error to later check whether the distance from the testing vector to projection region is not too long. Projecting the centered vector a_{test} onto each eigenface v_j produces its PCA coordinates z_{test}. We then perform a simple 1-nearest neighbor search: by scanning through all stored training projections z_i, we pick one whose Euclidean distance $\|z_{\text{test}} - z_i\|_2$ is smallest.

To avoid accidentally "recognizing" someone who is not in the database, we compare this minimum distance against a pre-determined threshold T_d and also the reconstruction error against the threshold T_r. If the closest match lies within that cutoff, we return its associated label; otherwise, we classify the face as Unknown.

Implementation and Results

Data Collection & Preprocessing

In our implementation, we worked with a dataset of face images. Each team member took around 25–30 training images of themselves to capture some variation in expression and lighting. Prior to applying PCA, all images were preprocessed to reduce unnecessary features of the photo: we converted them to grayscale, used a face detector "Haar Cascade" [9], which is an algorithm that can identify objects in images, irrespective of their scale in image and location. In enabled us to crop and align faces to a standard size. Each image was then resized to a fixed resolution (such as 100×100 pixels, i. e. $m = 10{,}000$ dimensions) and flattened into a vector.

Eigenface Computation

We set the number of principal components k based on the total variance explained. For our experiment this sufficient number was 20 as we observed that the eigenvalues drop off not so rapidly after this size of value. (Fig. 1)

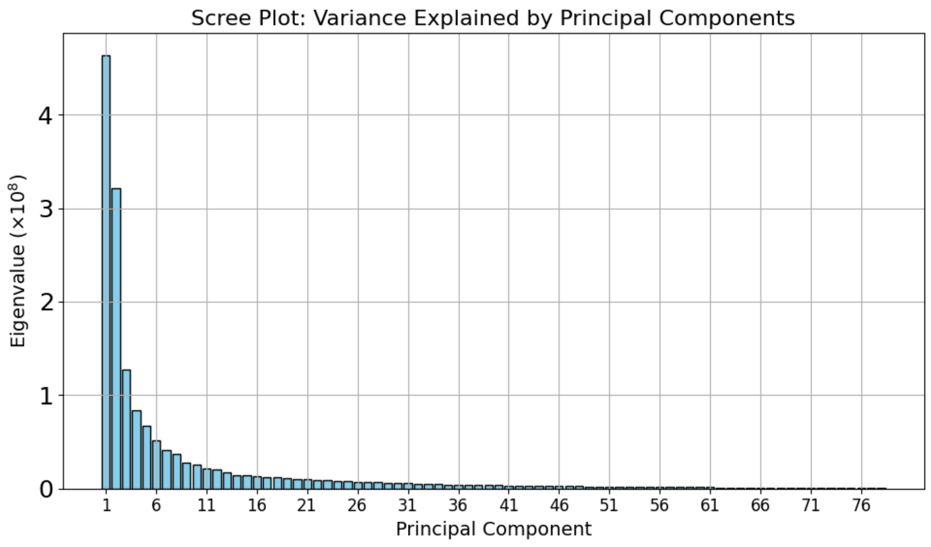

Figure 1: Change of Explained Variance with increasing index of Principal Components

Using the power method and deflation, we computed the top k eigenfaces (U). Then we changed the basis coordinates of our training images matrix by projecting it onto the eigenfaces ($Z = U^T \times X$).

Recognition and Threshold Tuning

During testing, each test image was projected to PCA space and we computed distances to all training feature vectors (Z). Initially, we set the threshold T_d (the T_r was constructed in a similar way), which was based on the conducted f1-score vs threshold value comparison. (Fig 2)

F1-score is an indicator that combines precision and recall into one number using a harmonic mean $F_1 = 2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$ Precision shows what proportion of the predicted positive answers are

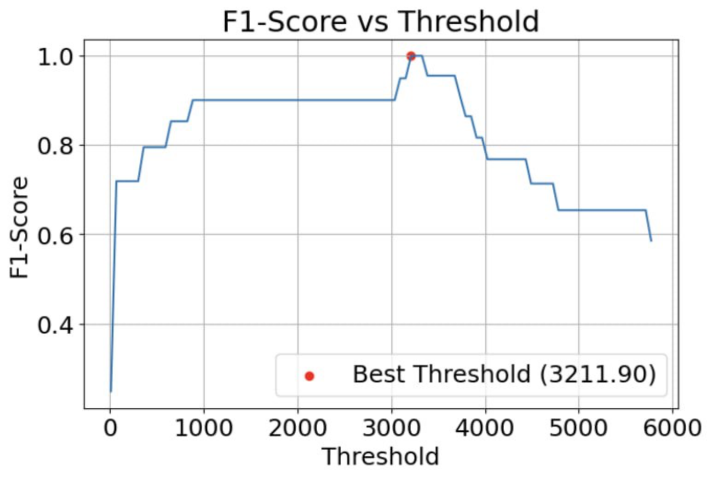

Figure 2: Picking the best threshold value using f1-score evaluations

correct, and recall shows what proportion of true positive cases were found. We chose the threshold at which the F1-score reaches a maximum (3211), as this provides the best balance between accuracy and completeness.

System Performance

The face recognition system was tested on a separate set of test images (Fig 3).

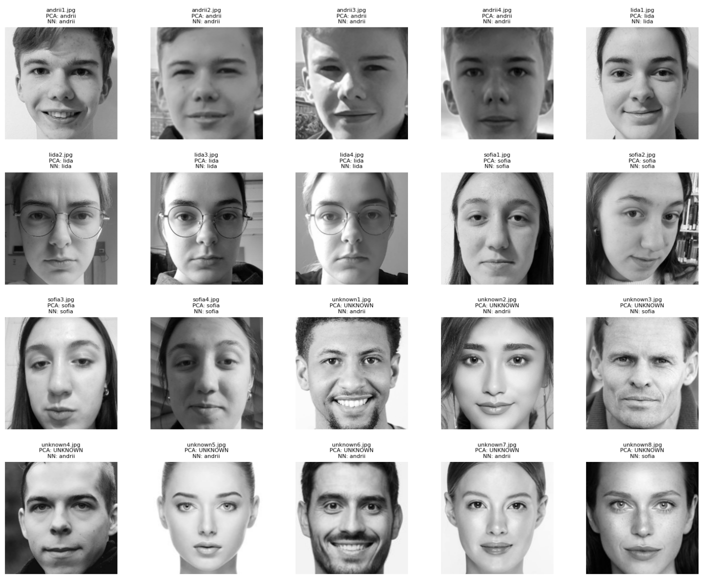

Figure 3: Testing the face recognition program on a set of 20 images (4 - of each team member, 8 - strangers)

In the project, a convolutional neural network [2] (CNN) was built to compare with our algorithm. The network consists of two convolutional layers with ReLU activation, each followed by max-pooling to reduce feature map dimensionality. After flattening, the data is processed by a dense layer with 128 neurons and ReLU activation, followed by a Dropout layer (0.3) to prevent overfitting, and finalized with a softmax layer outputting probabilities across three classes (andrii, lida, sofia).

We managed to achieve a much higher accuracy, compared to neural network's (Fig. 4). Importantly, when we presented an image of person not in the training set, the distance threshold mechanism correctly classified them as "Unknown" in most cases.

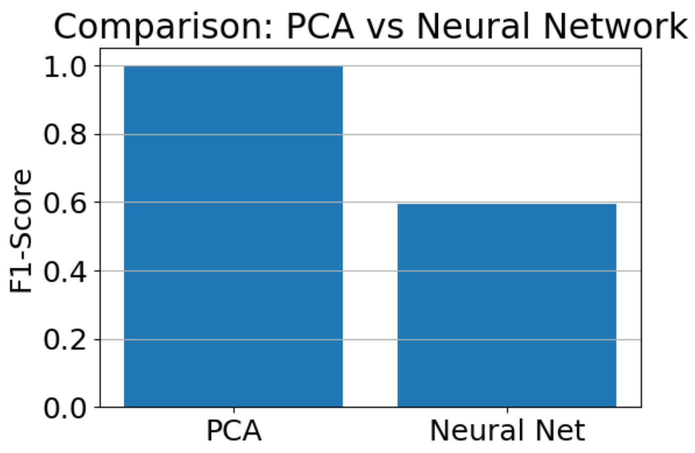

Figure 4: PCA Efficiency in terms of comparison to Neural Networks

Overall, the eigenface method provided a simple and effective solution for face recognition in our project. We were able to implement it in Python, using only basic linear algebra operations. The power method with deflation successfully extracted the principal component without needing a heavy eigen decomposition of the full covariance matrix. The nearest-neighbor classifier with threshold gave reasonable performance for identifying faces and chopping off the unknowns. The created datasets and the code of our project can be accessed on the GitHub repository[10].

References

[1] Manal Abdullah, Majda Wazzan, and Sahar Bosaeed. Optimizing face recognition using pca. *arXiv preprint arXiv:1206.1515*, 2012. URL https://arxiv.org/pdf/1206.1515.

[2] TensorFlow Authors. *Convolutional Neural Network (CNN) Tutorial*. TensorFlow, 2024. URL https://www.tensorflow.org/tutorials/images/cnn. Accessed: 2025-04-29.

[3] Ena Barčić, Petra Grd̦, and Igor Tomičić. Convolutional neural networks for face recognition: A systematic literature review. https://www.researchgate.net/publication/372322451_Convolutional_Neural_Networks_for_Face_Recognition_A_Systematic_Literature_Review, 2023. Accessed: 2025-04-28.

[4] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4th edition, 2013.

[5] Kyungnam Kim. *Face Recognition using Principal Component Analysis*. University of Maryland, 2000. Available at: http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf.

[6] M. Kirby L. Sirovich. Face recognition using principal component analysis. https://face-rec.org/interesting-papers/General/ld.pdf, 1986. Accessed: 1986-10-10.

[7] A. Pentland M. Turk. Eigenfaces for face recognition. https://www.face-rec.org/algorithms/PCA/jcn.pdf, 1991. Accessed: 1991-06-03.

[8] Vikas Maheshkar, Suneeta Agarwal, Vinay Kr. Srivastava, and Sushila Maheshkar. Face recognition using geometric measurements, directional edges and directional multiresolution information. *Procedia Technology*, 6:939–946, 2012. doi: 10.1016/j.protcy.2012.10.114. URL https://www.sciencedirect.com/science/article/pii/S2212017312006597.

[9] OpenCV.org. *Face detection using Haar cascades*, 2024. URL https://docs.opencv.org/4.x/d2/d99/tutorial_js_face_detection.html. Accessed: 2025-04-29.

[10] Sampara Sofia Semsichko Lidiya and Bilyi Andrii. face_recognition: Pca-based face recognition in python. https://github.com/LidaSemsichko/face_recognition.git, 2025. GitHub repository.

[11] Adrian Tam. Face recognition using principal component analysis. https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/, 2018. Accessed: 2025-04-28.